Expert Systems with Applications 39 (2012) 4532–4544

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Learning feature-projection based classifiers

Aynur Dayanik

Department of Computer Engineering, Bilkent University, Bilkent 06800, Ankara, Turkey

ARTICLE INFO

Keywords:
Classification learning
Inductive learning
Feature projections

ABSTRACT

This paper aims at designing better performing feature-projection based classification algorithms and presents two new such algorithms. These algorithms are batch supervised learning algorithms and represent induced classification knowledge as feature intervals. In both algorithms, each feature participates in the classification by giving real-valued votes to classes. The prediction for an unseen example is the class receiving the highest vote. The first algorithm, OFP.MC, learns on each feature pairwise disjoint intervals which minimize feature classification error. The second algorithm, GFP.MC, constructs feature intervals by greedily improving the feature classification error. The new algorithms are empirically evaluated on twenty datasets from the UCI repository and are compared with the existing feature-projection based classification algorithms (FIL.IF, VFI5, CFP, *k*-NNFP, and NBC). The experiments demonstrate that the OFP.MC algorithm outperforms other feature-projection based classification algorithms. The GFP.MC algorithm is slightly inferior to the OFP.MC algorithm, but, if it is used for datasets with large number of instances, then it reduces the space requirement of the OFP.MC algorithm. The new algorithms are insensitive to boundary noise unlike the other feature-projection based classification algorithms considered here.

1. Introduction

Classifier learning is one of the central problems in machine learning and data mining. Medical doctors, marketing professionals, curators of biological journal articles find rule-based classifiers more comprehensible because their domain knowledge becomes more transparent in those classifiers' data representation and decisions. Most practitioners already have some expectations about each feature's power to explain the variations in data. Therefore, they are more confident in using simple rules directly based on features, rather than on their complex combinations or transformations. Feature-projection based classifiers summarize labeled data separately on each feature dimension by means of histograms like Naive Bayes classifier or by pairwise disjoint intervals like FIL.IF, VFI5, and CFP, which are explained in Section 2, and classify new instances using these data representations. Therefore, those classifiers are simple but effective and intuitively appealing tools for the practitioners. In this paper, we aim at designing new feature-projection based classifiers which perform better than those previously proposed similar classifiers.

Some of the feature-projection based algorithms are feature interval learning (FIL.IF) (Dayanik, 2010), voting feature intervals (VFI5) (Demiröz & Güvenir, 1997; Güvenir, Demiröz, & Ilter, 1998; Güvenir & Emeksiz, 2000), *k*-nearest neighbor on feature projections (*k*-NNFP) (Akkuş & Güvenir, 1996), classification by feature partitioning (CFP) (Güvenir & Şirin, 1996), and Naive Bayes classifier (NBC) (Domingos & Pazzani, 1997; Mitchell, 1997; Witten & Frank, 2005), which are overviewed in Section 2. The *k*-NNFP algorithm uses feature projections to find the nearest observations on each feature dimension and then combines the predictions of the features to make final decision. The CFP, VFI5, and FIL.IF algorithms represent concept descriptions in the form of feature intervals, which are learned separately for each feature. The main difference among those algorithms is the way they construct the feature intervals. Naive Bayes classifier assumes that features are conditionally independent given the class, and uses the Bayes theorem to estimate the probability of each class given the feature values of the new example to be classified. Feature-projection based learning algorithms are simple, effective, efficient and robust to irrelevant features. They yield plausible concept descriptions, enable fast classification, do not require normalization of feature values, and handle missing feature values by simply neglecting them.

In this paper, we propose two new feature-projection based classification algorithms. They are batch supervised learning algorithms and represent the induced classification knowledge as a set of pairwise disjoint intervals on each feature dimension. In both algorithms, each feature participates in the classification by giving real-valued votes to classes. The prediction for an unseen example is the class receiving the highest vote. The first algorithm, OFP.MC

E-mail address: adayanik@cs.bilkent.edu.tr

0957-4174/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2011.09.133

(learning by optimal feature partitioning based on misclassification rates), learns feature intervals separately for each feature by optimally partitioning feature values to minimize the feature classification error. The second algorithm, GFP.MC (learning by greedy feature partitioning based on misclassification rates), constructs feature intervals in a greedy manner; it starts with point intervals and merges two consecutive intervals which results in the smallest immediate increase in the feature classification error, and repeats the latter until target number of intervals is achieved. The number of intervals for the final model is determined by comparing the feature classification errors for different number of intervals.

Both algorithms learn the partitions for every fixed number of intervals on some training set. True misclassification error is then estimated on a separate validation set. This is repeated five times—for each fold of 5-fold cross-validation—and the best number of intervals is found by a one-standard-error rule. OFP.MC learns the best partition on each training set using dynamic programming, while GFP.MC combines consecutive intervals greedily until the desired number of intervals is reached. In both algorithms, each feature participates in the classification by giving real-valued votes to classes. The prediction for an unseen example is the class receiving the highest vote.

In this paper, we propose two new classification algorithms using feature projections of the training data. On each feature dimension, classification knowledge is learned by partitioning features into pairwise disjoint intervals. Feature partitioning is related to discretization of linear (continuous) features (Dougherty, Kohavi, & Sahami, 1995; Liu, Hussain, Tan, & Dash, 2002; Yang & Webb, 2009). Discretizing a linear feature transforms it into a nominal (categorical) feature, where each value of the nominal feature corresponds to an interval of values of the linear feature. However, our focus is not on discretizing the linear features; we aim to develop better-performing feature-projection based classification algorithms.

The new algorithms are empirically evaluated on twenty datasets from the UCI repository and are compared with the existing feature-projection based classification algorithms (FIL.IF, VFI5, CFP, *k*-NNFP and NBC). The experiments demonstrate that the new algorithm OFP.MC outperforms other feature-projection based algorithms. The GFP.MC algorithm is slightly inferior to the OFP.MC algorithm, but, if it is used for datasets with large number of instances, then it reduces the space requirement of the OFP.MC algorithm. The new algorithms are also insensitive to boundary noise unlike the feature-projection based algorithms VFI5, CFP, FIL.IF, and NBC.

The next section starts with a review of the previous work on feature-projection based classification algorithms. Section 3 presents the optimal feature partitioning learning algorithm for classification. Optimal partitioning of linear features are described and illustrated on the *iris* dataset from the UCI repository in Section 4. Section 5 presents the greedy feature partitioning learning algorithm for classification. Section 6 presents the empirical evaluation of the algorithms on real-world datasets. The new algorithms are compared to the previously proposed feature-projection based algorithms. The complexity analysis of the algorithms is given in Section 7. Section 8 concludes with some remarks and suggestions for future research.

2. Previous work on feature-projection based algorithms

Feature-projection based classification algorithms represent the instances with their projections on feature dimensions. Some of the previously proposed such algorithms are feature interval learning (FIL.IF) (Dayanik, 2010), voting feature intervals (VFI5) (Demiröz & Güvenir, 1997; Güvenir et al., 1998; Güvenir & Emeksiz, 2000), *k*-nearest neighbor on feature projections (*k*-NNFP) (Akkuş & Güvenir, 1996), and classification by feature partitioning (CFP) (Güvenir & Şirin, 1996). These algorithms were shown to be fast, accurate, and robust to irrelevant features.

The *CFP algorithm* represents classification knowledge as sets of disjoint feature segments. It partitions the feature values into segments that are generalized or specialized as the training instances are processed. It constructs the segments incrementally, and the resulting concept descriptions are sensitive to the processing order of the training instances.

The *VFI5 algorithm* learns classification knowledge in batch mode and represents it as multi-class feature intervals. It finds on each linear feature and for each class the class endpoints, which are the lowest and highest feature values with the same class labels. Pairwise disjoint intervals are constructed at every endpoint and between every pair of the closest endpoints of possibly different classes. If the feature is nominal, then every distinct point forms a point interval. The VFI5 algorithm has been implemented in the WEKA machine learning workbench (Hall et al., 2009), and has been used in several medical applications (Güvenir et al., 1998; Güvenir & Emeksiz, 2000).

The *FIL.IF algorithm* is a batch supervised learning algorithm, which learns the concept descriptions in the form of a set of disjoint intervals separately for each feature. On every linear feature, a point interval is constructed at each observed feature value. The algorithm then generalizes the point intervals into the range intervals by merging all of the neighboring single-class point intervals with the same class labels. However, multi-class point intervals are left alone, and at the end of the training process, they are converted to single-class point intervals by selecting the classes with the highest relative representativeness values as their class labels. The feature intervals are always disjoint. Nominal features have only (possibly multi-class) point intervals.

The *k*-NNFP algorithm represents instances as separate collections of feature projections. Given a new example, it uses feature projections to find the nearest observations on each feature dimension. Each feature has exactly *k* votes and distributes them between the classes of the *k*-nearest training observations. The votes of the individual features are summed; the class with the highest vote is predicted to be the class of the new example.

All feature-projection based algorithms predict the class of a new example as the label with the highest total weighted votes of the individual features. These algorithms allow faster classification than other instance-based learning algorithms because separate feature projections can be organized for faster classification. These algorithms easily and naturally handle the missing feature values by neglecting the class votes from the missing features. The major drawback of these algorithms is that the descriptions involving a conjunction between two or more features cannot be represented. Despite of that, they are still reported to be successful on real-world datasets (Akkuş & Güvenir, 1996; Dayanik, 2010; Güvenir et al., 1998; Güvenir & Şirin, 1996).

Bayesian classifiers from pattern recognition are based on probabilistic approaches to inductive learning in statistical concept learning tasks. The method estimates the posterior class probability of an instance given its observed feature values. The class is predicted as the label with the highest estimated posterior probability (Duda, Hart, & Stork, 2000; Fukunaga, 1990). Bayesian classifiers assume in general that features are not statistically independent unlike Naive Bayesian classifier (NBC) (Mitchell, 1997; Witten & Frank, 2005). NBC has been successfully used in a wide variety of problem domains (Chandra & Gupta, 2011; Chena, Huanga, Tiana, & Qua, 2009; Hsu, Huang, & Chang, 2008; Kim, Han, Rim, & Myaeng, 2006; Lewis, 1998), and has been shown to perform well in many domains even when the independence assumption is violated (Domingos & Pazzani, 1997). Domingos and Pazzani establish the necessary and sufficient conditions for the optimality of NBC under zero-one loss (Domingos & Pazzani, 1997).

3. OFP.MC: Optimal feature partitioning learning algorithm for classification

Here we describe the new feature-projection based algorithm OFP.MC, which generalizes the class information on every feature dimension in the form of pairwise disjoint intervals.

The algorithm aims at capturing the relation between the class labels and feature values with the best possible choices of the intervals. Therefore, the feature partition intervals are chosen so as to minimize the total number of misclassifications on the training dataset.

In order to prevent the overfitting, the set of labeled examples is divided into training and validation sets. Optimal feature partitions are learned from the training set, but their unknown true misclassification rates on the unseen data are estimated from the test results on the validation set.

In order to reduce the potential biases possibly introduced with the arbitrary division of the labeled examples into training and validation sets, the average misclassification rates and their standard errors are recalculated by a fivefold cross-validation. The final choices of the optimal number of feature intervals are determined with a one-standard-error rule. The optimal feature partitions with the optimal number of feature intervals are finally found by minimizing the total number of misclassifications on the entire set of labeled examples. For every feature interval in the optimal feature partition, we calculate the end-points, the class weights, and the class prediction, all of which are needed for classification of new examples.

The independent optimizations of the number and breakpoints of the feature intervals balance the trade-off between model bias and model variance. On the one hand, the optimal choice of the interval breakpoints on the training data explains the labeled examples in the best possible way and reduces the final model's bias. On the other hand, the optimal choice of the number of intervals on the validation set eliminates the unnecessarily complex models and reduces the final model's variance.

The classification with OFP.MC is simple and fast. For each nonmissing feature value of a given test example, the interval into which the feature value falls is identified. Then the class weights of those intervals are summed, and the class with the largest total weight is predicted. The training and classification algorithms are precisely stated in Figs. 1 and 2, respectively. The numerical results and comparisons to other feature-projection based learning algorithms are discussed in Section 6.

In the next section, the details of the training in OFP.MC algorithm are explained, and both its training and classification processes are illustrated on the *iris* dataset from the UCI repository.

4. Optimal partitioning of linear features in the training of OFP.MC

Suppose that a set of labeled examples with several features is given. The examples have exactly one of L distinct class labels. For every fixed feature, we solve the following optimal feature partitioning problem.

Suppose that N distinct feature values $b_1 < \cdots < b_N$ are observed, at each of which there may be multiple instances with the same and/or different class labels. The *optimal feature partitioning problem* is

(i) to attach class labels to $b_1, \ldots, b_N$ so as to minimize the total number of misclassified examples on the same feature, with at most k label switchings along the same feature dimension, and at the same time,
(ii) to choose the maximum number of label switchings $0 \leqslant k \leqslant N-1$ in order to minimize the generalization error.

The consecutive feature values with the same labels are later grouped together to form at most $k+1$ pairwise disjoint feature intervals.

This problem is solved in two stages. In the first stage, for every fixed integer $0 \leqslant k \leqslant N-1$, an optimal labeling of the feature values with at most k label switchings is found on some training data. The misclassification rate of each labeling is then evaluated on an independent validation set.

While the misclassification rate on the same training set is always a nonincreasing function of k, it typically decreases first and increases later with k on the validation data. To avoid over-fitting to data, optimal k is chosen based on the misclassification rates estimated on the validation set.

The sampling errors in the estimates of the misclassification rates are likely to obscure the real differences between the unobserved true misclassification rates for different k values. In order to reduce the sampling variances of those estimates on the final choice of optimal maximum number of switchings k, both the optimal labeling of feature values with at most k label switchings and the independent assessment of its misclassification rate for each $0 \leqslant k \leqslant N-1$ are repeated on several different disjoint training and validation sets, which are obtained in this paper from applying fivefold cross-validation to a given set of labeled examples.

For every $0 \leqslant k \leqslant N-1$, the sample average and sample standard deviation of five misclassification rates are calculated. The average misclassification rates are plotted against k, and its minimum $k_{\min}$ is found. In general, the sampling errors may make the misclassification rates for the values of k near $k_{\min}$ look different, but this difference is often statistically insignificant. We therefore set our final choice k_{opt} of maximum number of label switchings to the smallest of all k values with an average misclassification rate less than the (minimum) average misclassification rate plus its standard error at k_{opt}. The main steps of the outlined method will now be given in more details.

4.1. Optimal feature partitioning by using dynamic programming

On every fixed feature of a given training data, optimal labeling of N distinct feature values $b_1 < \cdots < b_N$ with at most k label switchings can be found simultaneously for all $0 \leqslant k \leqslant N-1$ by using dynamic programming. Let us add an artificial point $b_{N+1} > b_N$ with no actual labeled data. For $n = 1, \ldots, N$, $\ell = 1, \ldots, L$, and $k = 0, 1, \ldots, N-1$, let us define the value function

$$v(n,\ell,k) = \text{minimum total number of misclassified examples on } \{b_1,\ldots,b_n\} \text{ obtainable by a labeling of } b_1,\ldots,b_n \text{ if } b_{n+1} \text{ already has label } \ell \text{ and at most k abel switchings are allowed on } \{b_1,\ldots,b_n\}$$

and the misclassification function

$$e(\ell,n) = \text{the number of misclassified examples at } b_n \text{ if every example with feature value } b_n \text{ is assigned label } \ell.$$

The value function satisfies the boundary conditions

$$v(1,\ell,k) = \left\{ \begin{array}{ll} e(\ell,1), & \text{if } k=0 \\ \min\limits_{1\leqslant\tilde{\ell}\leqslant L} e(\tilde{\ell},1), & \text{if } k>0 \end{array} \right\} \quad \text{for every } \ell = 1,\ldots,L$$

and $k = 0, 1, \ldots, N-1$.

The minimum number of misclassifications achievable on the training set by any labeling of the feature values with at most k label switchings equals

```
train(Training Set)
begin
    for each fold of 5-fold cross-validation on training set
        divide training set into train and validation data
        for each linear feature f
            sort train data on feature f
            partition feature f optimally using dynamic programming (DP), compute
                    value function (minimum number of misclassified examples) and
                    policy function (optimal first labels of feature values )
            for each k from 0 to the number of distinct feature values on feature f
                find optimal labels on feature f from value and policy functions ...
                                                    using at most k breakpoints
                construct intervals on sorted train data using optimal labels
                evaluate intervals on feature f using validation data, ...
                                                    compute misclassification error
    for each feature f
        if f is linear then
            choose optimal number of breakpoints k_opt by analyzing the performance curve
                    (average misclassification error from 5-fold cross-validation versus k)
            sort entire training set on feature f
            partition f using DP, compute value and policy functions
            find optimal class labels for feature values from value and policy functions ...
                                                    using at most k_opt breakpoints
            construct intervals on sorted train data using optimal class labels
        else
            /* if f is a nominal feature, no generalization is done */
            each distinct point forms a point interval
        compute-interval-classweights(intervals)
end.
compute-interval-classweights(intervals)
begin
    let class.count[c] be the number of instances with class c in training set
    for each interval
        for each class c
            compute interval.count[c], the number of feature values with class c in the interval
            interval.classweight[c] = interval.count[c]/class.count[c]
        normalize-interval-classweights so that ∑_c interval.classweight[c] = 1
end.
```

Fig. 1. The training process of the OFP.MC algorithm.

$\min_{1\leqslant \ell \leqslant L} v(N,\ell,k)$ for every $k = 0,1,\ldots,N-1$,

and we can find $v(N,\ell,k)$ for every $\ell = 1,\ldots,L$ and $k = 0,1,\ldots,N-1$ by recursively calculating the optimality equations

$$v(n,\ell,k) = \min\left\{ e(\ell,n) + v(n-1,\ell,k), \min_{1\leqslant \tilde{\ell}\leqslant L, \tilde{\ell}\neq \ell} [e(\tilde{\ell},n) + v(n-1,\tilde{\ell},k-1)] \right\}$$

for every $n = 1,\ldots,N$, $\ell = 1,\ldots,L$, $k = 0,1,\ldots,N-1$; see Fig. 3. The optimal label for b_n when at most k label switchings is left for labeling the feature values in $\{b_1,\ldots,b_n\}$ and b_{n+1} already has label ℓ is given by

$$p(n,\ell,k) = \begin{cases} \ell, & v(n,\ell,k) - v(n-1,\ell,k) = e(\ell,n), \\ \arg\min_{1\leqslant \tilde{\ell}\leqslant L, \tilde{\ell}\neq \ell} [e(\tilde{\ell},n) + v(n-1,\tilde{\ell},k-1)], & v(n,\ell,k) - v(n-1,\ell,k) < e(\ell,n). \end{cases}$$

```
classify(test)
begin
    for each class c
        vote[c] = 0
    for each feature f
        if feature value test_f of test is known
            interval = search-interval(f, test_f)
            /* each feature contributes for each class proportional to its weight */
            for each class c
                feature.vote[c] = interval.classweight[c]
                vote[c] = vote[c] + feature.vote[c] × weight[f]
    return the class c with the highest vote[c]
end.
search-interval(f, value)
begin
    return interval containing value on feature f
end.
```

Fig. 2. The classification process of the OFP.MC algorithm.

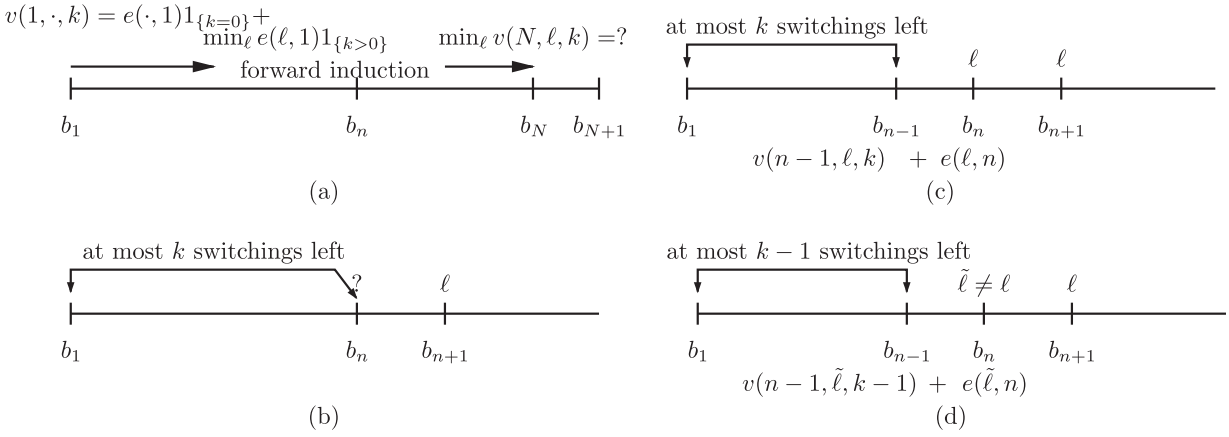

Fig. 3. The dynamic programming algorithm: we start in (a) with known boundary condition and proceed forward to determine the optimal labels of the feature values. As we find in (b) the optimal labeling of b_n when b_{n+1} has label ℓ and at most k label switchings are left for $\{b_1,\ldots,b_n\}$, we must compare the minimum remaining number of misclassifications under the two alternative decisions depicted by (c) and (d).

For every fixed k, optimal labels of $b_1,\ldots,b_N$ can now be determined in backwards direction by means of the policy function $p(\cdot,\cdot,\cdot)$. Let

$$k_{N+1} = k \quad \text{and} \quad \ell_{N+1} = \arg\min_{1\leqslant\ell\leqslant L} v(N,\ell,k_{N+1}).$$

After the optimal labels of $b_{N+1}, b_N, \ldots, b_{n+1}$ are found, then an optimal label of b_n and the maximum remaining number of label switchings are, respectively, given by

$$\ell_n = p(n,\ell_{n+1},k_{n+1}) \quad \text{and} \quad k_n = k_{n+1} - 1_{(\ell_n \neq \ell_{n+1})} \quad \text{for every } 1 \leqslant n \leqslant N,$$

where $1_{(a\neq b)}$ equals 1 if $a \neq b$ and 0 otherwise.

Let us emphasize that optimal labeling $\ell_1,\ldots,\ell_N$ of feature values, respectively, $b_1,\ldots,b_N$ with "at most k label switchings" will contain strictly less than k label switchings if this gives statistically same or strictly smaller overall misclassification rate than that of exactly k label switchings. The dynamic programming formulation favors the smallest possible number of label switchings (and number of intervals) on each feature dimension among all choices which have statistically similar powers to explain the labeled examples.

4.2. The construction of optimal feature intervals corresponding to maximum k label switchings

After optimal labels of feature values are found for any given fixed maximum number of label switchings k, we collapse all consecutive feature values with identical class labels into one and the same interval and assign to the interval the same common class

label of all member feature values. The left (respectively, right) boundary of the leftmost (respectively, rightmost) interval is set to minus (respectively, plus) infinity. The consecutive intervals are extended halfway toward each other with the midpoint included at random to one of those intervals.

For each interval, we record its *left-* and *right-endpoints*, its *class prediction*, and for each class its *class weight*, which is calculated as the fraction of all examples of that class in the union of training and validation sets which fall into that interval. The interval's class prediction is a class label with the highest class weight of the interval.

4.3. The performance evaluation of optimal feature intervals corresponding to maximum k label switchings

For every maximum k number of label switchings, the dynamic programming solution readily provides the minimum misclassification rate

$$\frac{\min_{1\leqslant \ell\leqslant L} v(N,\ell,k)}{\text{number of training examples}}$$

on the training set, but this is typically an optimistic estimate of the misclassification rate of the model on unseen examples. A more realistic estimate is obtained on an independent validation set, and it is calculated as the fraction of the examples of the validation set which have different class labels than *class predictions* of the feature intervals into which they fall.

4.4. The calculation of the performance curve and optimal choice of the maximum number of label switchings

A given set of labeled examples is divided into five approximately equal parts. Each part is used exactly once as a validation set, while the union of remaining four parts is used as the training set.

For each maximum number of label switchings k, optimal feature intervals are learned from the training data and the performance (the misclassification rate) is calculated on the validation set. This is repeated five times, giving five estimates of the misclassification rate of optimal feature intervals corresponding to each k.

The average misclassification rate and its standard error are calculated for every k. The minimum average misclassification rate is attained at some $k_{\min}$ value. The final choice k_{opt} of the maximum number of label switchings is set to the smallest of all those k values having average misclassification rate less than or equal to the sum of the average misclassification rate and its standard error both corresponding to $k_{\min}$.

4.5. The construction of the final feature intervals for classification

After the final maximum number of label switchings k_{opt} is determined, we use the dynamic programming on the entire set of labeled examples to find the optimal labeling of the feature values with at most k_{opt} label switchings, from which we construct the feature intervals. For each of those intervals we determine left- and right-endpoints, class prediction, and class weights as described earlier.

4.6. Illustration on the iris dataset

The optimal partitioning of linear features is central to the training process of the new classification algorithm, which was listed in Fig. 1. Here we shall illustrate it on the *iris* dataset from the UCI repository (Asuncion & Newman, 2007). Fig. 4 displays the labeled examples projected on each of four feature dimensions.

The average misclassification rates and their standard errors corresponding to optimal feature partitions with at most k label switchings are calculated for each feature by a fivefold cross-validation. The second row in Fig. 5 presents the plots of the average misclassification rates and their one-standard-error confidence bounds for each value of k calculated on the validation sets. On every feature, we first find the maximum number of label switchings $k_{\min}$ at which the average misclassification rate is minimized. On each figure, $k_{\min}$ is marked by a vertical dashed line. The horizontal dashed line goes through the upper one-standard-error confidence bound of the average misclassification rate at $k_{\min}$. The final maximum number of label switchings k_{opt} is the smallest k value whose average misclassification rate is below the dashed horizontal one-standard-error line; its place is marked by a solid vertical line in each plot.

Evident from Fig. 4, the labeled examples are more separable on features 3 and 4 than on 1 and 2. Therefore, the average misclassification rates plotted in the second row of Fig. 5 decrease first and increase later on features 1 and 2. The standard errors of the average misclassification rates are also larger on features 1 and 2 than on 3 and 4. The optimal k_{opt} value coincides with the minimum $k_{\min}$ value on features 1, 3, and 4, but k_{opt} is strictly less than $k_{\min}$ on feature 2. More precisely, the optimal number of label switchings k_{opt} are 2, 1, 2, and 2 for features 1, 2, 3, and 4, respectively. By using the entire set of labeled examples and the dynamic programming algorithm, those features are partitioned optimally into at most $k_{\text{opt}}+1$, namely, 3, 2, 3, and 3 pairwise disjoint intervals, respectively. The second row in Fig. 6 shows the optimal partitioning of the features into intervals. The vertical solid lines mark the boundaries of the intervals, and the class labels under the horizontal axes indicate the class predictions of the intervals. For each feature interval, the class weights are calculated as in Table 1 and visually displayed by the barplots of Fig. 7.

The first rows of Figs. 5 and 6 display, respectively, the misclassification rates calculated on the *training sets* and the feature partitions found with the one-standard-error rule applied to the rates just described. The misclassification rates calculated in the first row of Fig. 5 on training sets are optimistic estimates of the misclassification rates of the OFP.MC learning algorithm on the unseen examples. These estimates are lower and have smaller confidence intervals than those calculated in the second row of Fig. 5 from the validation sets. Moreover, the plots in the first row of Fig. 5 are always nonincreasing. Therefore, the one-standard-error rule typically returns large values of k_{opt} and suggests that the features be partitioned into large number of intervals. However, when the corresponding optimal feature intervals are plotted in the first row of Fig. 6, one realizes that the additional intervals do not appear to capture any new and meaningful patterns present in the labeled examples. In other words, the number of intervals found with the one-standard-error rule applied to misclassification rates calculated on the training set leads to a model which severely overfits to the data.

Let us next illustrate how the model of feature intervals classifies the test example

$$\langle (7.0, 3.2, 4.7, 1.4), 1\rangle,$$

which has feature values 7.0, 3.2, 4.7, 1.4 on features 1, 2, 3, 4, respectively, and true class label 1. We start by finding on each feature an interval which contains the feature value of the test example. We then collect the class weights of the intervals and sum the weights for each class. The class with the largest total weight is predicted. Table 2 shows the details of the calculations. While features 1 and 2 predict classes 2 and 0, respectively, features 3 and 4 predict class 1. In the end the sum 2.18 of the weights of all four features decisively predict Class 0, which also turns out to be the true class label of the test instance.

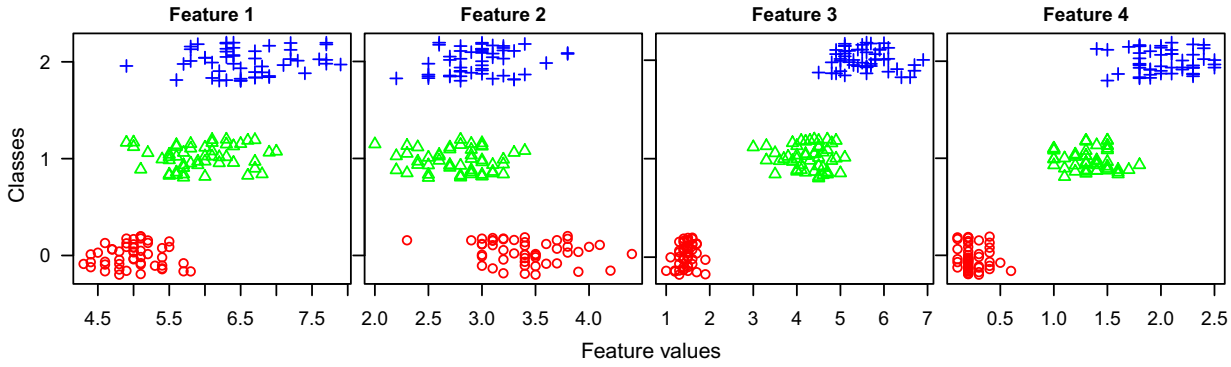

Fig. 4. Iris dataset. There are three classes 0, 1, and 2, which are jittered in order to identify distinct examples with the same feature values.

Table 1
Class weights learned by OFP.MC algorithm for *iris* dataset.

	Feature 1			Feature 2		
Intervals	$(-\infty, 5.45]$	$(5.45, 6.15]$	$(6.15, \infty)$	$(-\infty, 3.05]$	$(3.05, \infty)$	
Class 0	0.91	0.08	0	0.09	0.61	
Class 1	0.09	0.70	0.26	0.52	0.12	
Class 2	0	0.21	0.74	0.39	0.27	
	Feature 3			Feature 4		
Intervals	$(-\infty, 2.60]$	$(2.60, 4.95]$	$(4.95, \infty)$	$(-\infty, 0.80]$	$(0.80, 1.75]$	$(1.75, \infty)$
Class 0	1	0	0	1	0	0
Class 1	0	0.90	0.05	0	0.90	0.03
Class 2	0	0.10	0.95	0	0.10	0.97

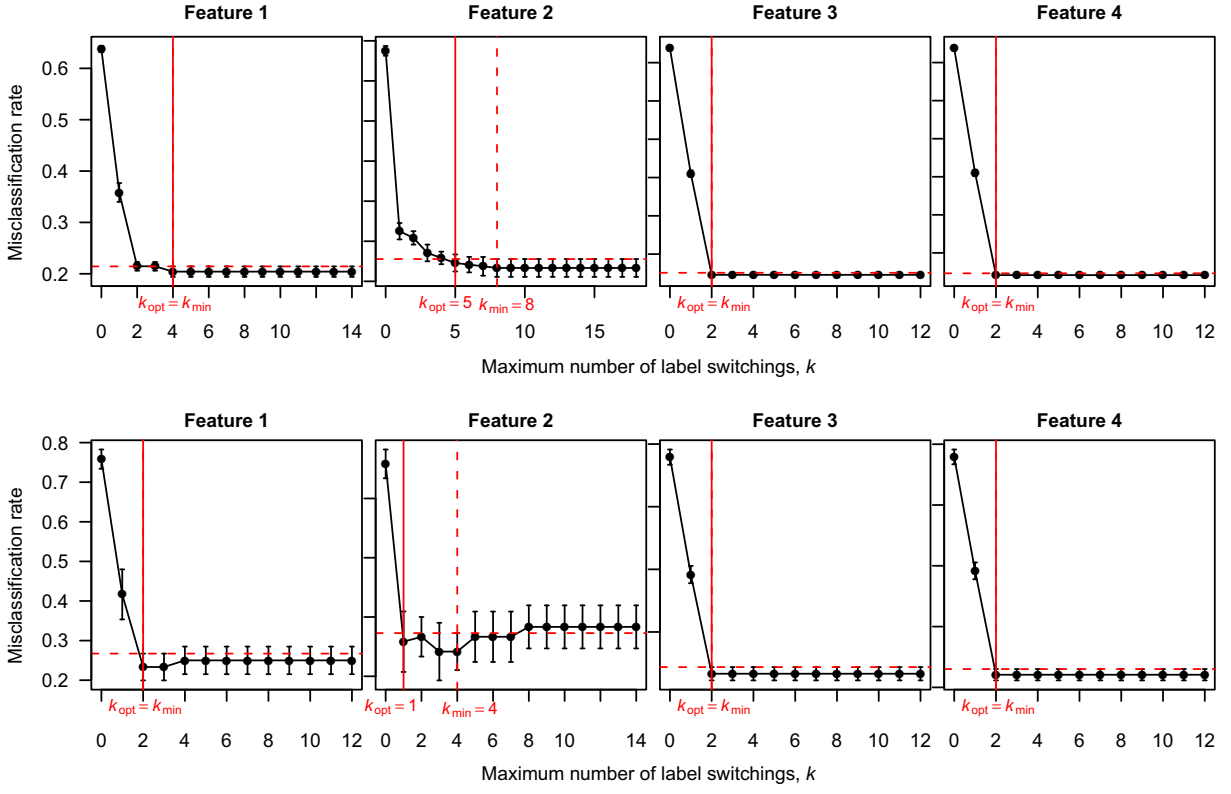

Fig. 5. The curves of average misclassification rates and their one standard error confidence intervals calculated on training sets, above, and validation sets, below.

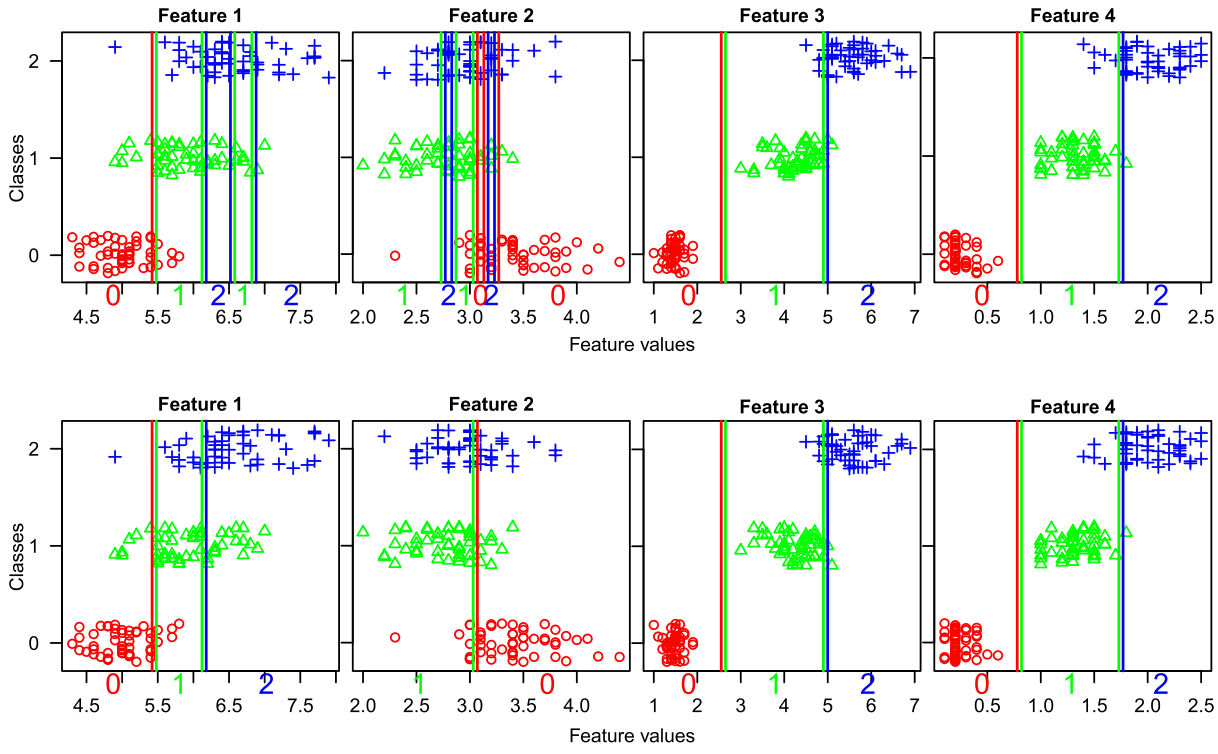

Fig. 6. Optimal feature partitions using misclassification rates on training sets, above, and validation sets, below. Interval boundaries and class predictions are shown. Additional intervals found above do not seem to capture new meaningful patterns.

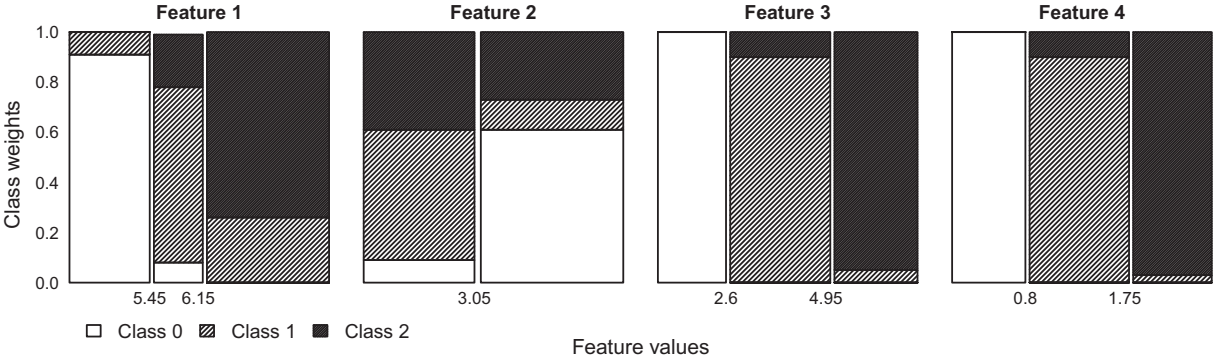

Fig. 7. The barplots of the feature interval class weights reported in Table 1 for *iris* dataset. The bar heights of different colors equal the interval class weights of the corresponding classes. Features 3 and 4 seem to separate the labeled examples better than features 1 and 2.

5. GFP.MC: greedy feature partitioning learning algorithm for classification

For the classification problems with a lot of training instances, optimal partitioning of the features into disjoint intervals may require prohibitively large storage space for the value and policy functions.

Table 2
Illustration of the classification of the test example $\langle(7.0, 3.2, 4.7, 1.4), 1\rangle$. Bold numbers highlight the largest class weights on each feature and the overall weight of the winning class.

	Feature 1	Feature 2	Feature 3	Feature 4		
Value	7.0	3.2	4.7	1.4		
Interval	$(6.15, \infty)$	$(3.05, \infty)$	$(2.60, 4.95]$	$(0.80, 1.75]$		
	Class weights				Sum	Prediction
Class 0	0	**0.61**	0	0	0.61	
Class 1	0.26	0.12	**0.90**	**0.90**	**2.18**	**Class 1**
Class 2	**0.74**	0.27	0.10	0.10	1.21	

Here we would like to describe a greedy feature partitioning learning algorithm, GFP.MC, which requires significantly less storage space. The algorithm is suboptimal, as an example below shows, in the sense that it does not always find a partition with the smallest possible number of misclassifications on the training set, but still competes on the numerical examples with the optimal feature partitioning learning algorithm OFP.MC; see Table 4 and Fig. 10 of Section 6.

In the training phase and on every feature dimension, the greedy algorithm initially forms one point interval at every feature value and assigns to it the label of the majority class at that feature value. Then neighboring intervals with the same interval class labels are merged. The number of feature intervals and the number of misclassifications on the independent validation set are counted and stored.

There are always two neighboring intervals such that merging and relabeling them with the label of the majority class in the new interval increases the total misclassifications on the training data by the least possible number. We find any of those two neighboring intervals, merge them, and assign to the new interval the label of the majority class in that interval. The number of feature

```
train(Training Set)
begin
    for each fold of 5-fold cross-validation on training set
        divide training set into train and validation data
        for each linear feature f
            sort train data on feature f
            partition feature f using greedy approach:
                    each distinct point forms a point interval
                    merge neighboring intervals with same class label into range intervals
            extend intervals to midpoints
            for each k from the number of intervals on feature f to 0
                mergeBestTwoIntervals(intervals)
                merge neighboring intervals with same class labels
                evaluate intervals on feature f using validation data, ...
                                                    compute misclassification error
    for each feature f
        if f is linear then
          sort entire training set on feature f
          choose optimal number of intervals k_opt by analyzing the performance curve
                (average misclassification error from 5-fold cross-validation versus k)
          partition f using greedy approach:
                    each distinct point forms a point interval
                    merge neighboring intervals with same class labels
          extend intervals to midpoints
          while (number of intervals > k_opt)
                MergeBestTwoIntervals(intervals)
                merge neighboring intervals with same class labels
        else
          /* if f is a nominal feature, no generalization is done */
          each distinct point forms a point interval
        compute-interval-classweights(intervals) /* from Figure 1 */
end.
mergeBestTwoIntervals(intervals)
begin
    for each interval in intervals
        compute the change in error if it is merged with the interval on the right
    pick the interval and its neighbor on the right giving the smallest increase in error, and merge them
end.
```

Fig. 8. The training process of the GFP.MC algorithm.

intervals and the number of misclassifications on the independent validation set are counted and stored. This procedure is repeated as long as there is at least one interval which can be eliminated.

The entire procedure is repeated five times with fivefold cross-validation. The average number of misclassifications and their standard errors are calculated for each maximum number of feature intervals. The greedy selection of the number of intervals for the final model is found by using the same one-standard-error rule described for the optimal feature partitioning algorithm OFP.MC in Section 4.4. The greedy partitioning algorithm is then run on the entire set of labeled examples until we find the first partition with the (largest) number of intervals less than or equal to the number found by the one-standard-error rule. For each interval in this final partition, the endpoints are extended, the class weights are calculated, and the interval predictions are determined as in exactly the same way as it was done for the optimal feature partitioning learning algorithm in Section 4.2. The procedure is repeated for every feature. The training process of the GFP.MC algorithm is given in Fig. 8. The classification process is the same as that of the OFP.MC algorithm; see Fig. 2.

A counterexample showing that GFP.MC is suboptimal. Suppose that eight labeled examples have feature values and class labels, respectively, as in

$$\langle 1, C_1\rangle,\ \langle 2, C_1\rangle,\ \langle 3, C_0\rangle,\ \langle 4, C_0\rangle,\ \langle 5, C_1\rangle,\ \langle 5, C_1\rangle,\ \langle 5, C_1\rangle,\ \langle 6, C_0\rangle,$$

after the examples are projected on some fixed feature dimension. In Fig. 9, every feature interval is shown by a box, below and above which its class label and the number of examples from different classes in it are stated. The feature intervals with class labels C_1 and C_0 have, respectively, dark and light colors. The greedy partition with at most two pairwise disjoint intervals makes three misclassifications, while there is a partition which makes two misclassifications with two pairwise disjoint intervals. This example illustrates that the greedy algorithm may return a suboptimal feature partition which makes more misclassifications on the training set than the minimum achievable with at most the same number of feature

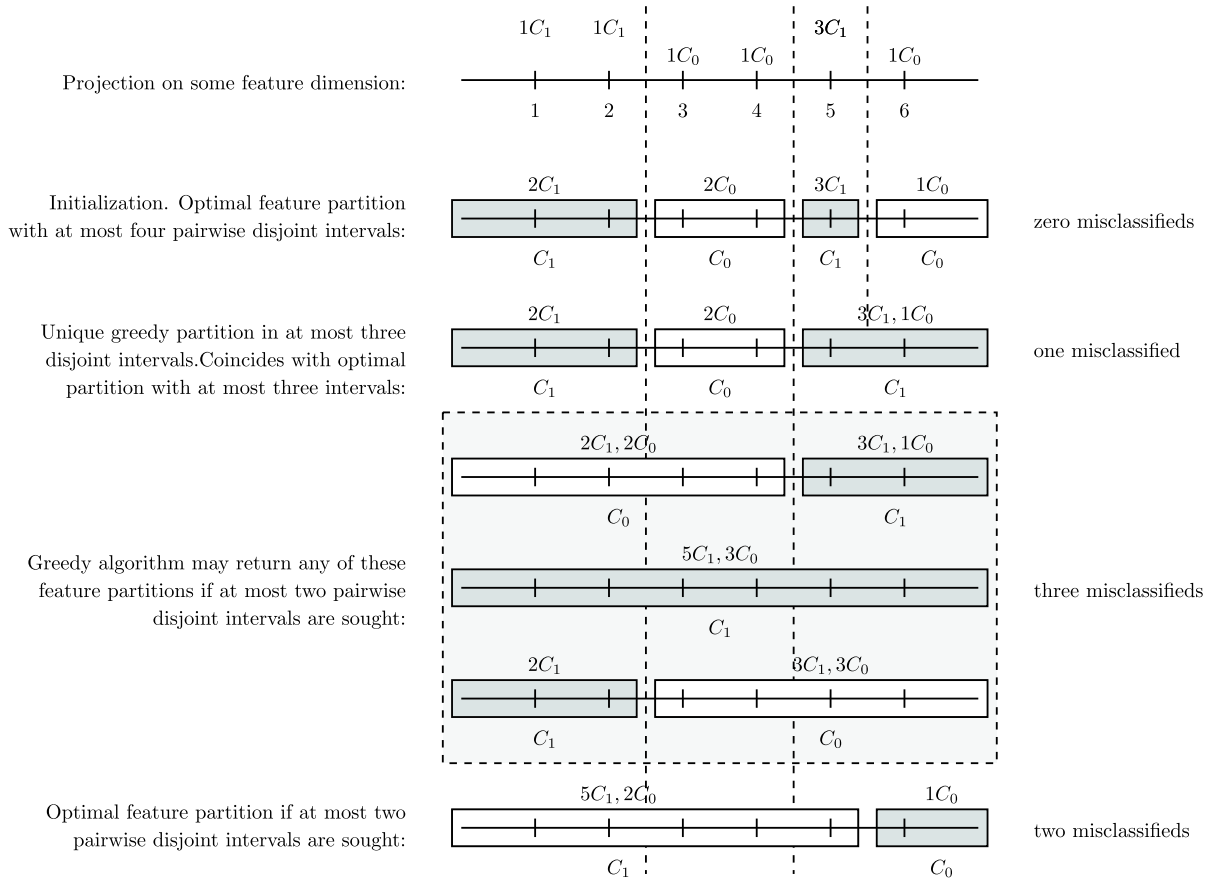

Fig. 9. The greedy feature partitioning learning algorithm GFP.MC can be strictly suboptimal in the sense that it does not always find a partition with the least possible number of misclassifications on the training set. In the example above, every greedy partition with at most two intervals makes three misclassfications, while at least one partition with two intervals makes two misclassifications on the training data.

Table 3
Properties of twenty real-world datasets from UCI-Repository.

Dataset	Size	# Of features	# Of linear features	# Of classes	Baseline accuracy (%)
arrhythmia	452	279	272	16	54
bcancerw	699	10	10	2	66
cleveland	303	13	6	2	54
dermatology	366	34	33	6	31
echocardio	74	7	6	2	68
glass	214	9	9	6	36
haberman	306	3	3	2	74
heart	270	13	7	2	56
horse	368	22	7	2	63
hungarian	294	13	6	2	64
ionosphere	351	34	34	2	64
iris	150	4	4	3	33
mammog.-mas.	961	5	3	2	54
new-thyroid	215	5	5	3	70
parkinsons	195	22	22	2	75
segmentation	2310	19	19	7	14
sonar	208	60	60	2	53
vehicle	846	18	18	4	26
wine	178	13	13	2	40
yeast	1484	8	8	10	31

intervals. In this sense, the greedy feature partitioning learning algorithm should not be expected to be optimal.

6. Experimental results

This section presents empirical evaluations of the OFP.MC and GFP.MC algorithms. The algorithms are compared to previously proposed feature-projection based algorithms, FIL.IF, VFI5, CFP, *k*-NNFP, and NBC.

All of those algorithms are evaluated on twenty datasets from the UCI machine learning repository (Asuncion & Newman, 2007). Table 3 describes each dataset, including the number of instances, features, linear features, classes, and the baseline classification accuracy, which is obtained by predicting the class of every test instance with the most frequent class in the dataset.

Table 4
The average percentage accuracy results of OFP.MC, GFP.MC, FIL.IF, VFI5, CFP, *k*-NNFP, and NBC algorithms. For each data set, the accuracy of the best performing algorithm is shown in boldface.

Dataset	OFP.MC	GFP.MC	FIL.IF	CFP	VFI5	*k*-NNFP	NBC
arrhythmia	**65.27**	59.20	56.38^{3+}	54.22^{3+}	61.60^{1+}	54.22^{3+}	58.06^{3+}
bcancerw	96.57	96.57	96.42	95.71^{3+}	95.19^{3+}	95.42^{2+}	$\mathbf{97.40}^{2-}$
cleveland	82.12	82.98	$\mathbf{83.58}^{2-}$	77.70^{3+}	82.12	81.46	79.94^{2+}
dermatology	95.19	95.46	63.77^{3+}	50.00^{3+}	$\mathbf{96.72}^{2-}$	52.24^{3+}	77.93^{3+}
echocardio	**71.90**	69.70	67.26^{2+}	69.43^{1+}	70.30	68.10^{1+}	70.59
glass	**64.37**	61.86	54.66^{2+}	48.03^{3+}	60.55^{1+}	60.48	53.92^{3+}
haberman	71.18	72.68	72.75	$\mathbf{74.12}^{1-}$	59.48^{3+}	73.53	70.98
heart	**82.52**	82.37	80.74	76.67^{3+}	81.11	79.56^{2+}	79.77^{3+}
horse	78.75	77.93^{1+}	77.19^{3+}	66.76^{3+}	78.64	69.58^{3+}	$\mathbf{80.27}^{3-}$
hungarian	82.92	83.61	84.15	68.71^{3+}	$\mathbf{84.96}^{1-}$	74.97^{3+}	82.58
ionosphere	89.23	89.35	$\mathbf{92.53}^{3-}$	86.78^{2+}	80.97^{3+}	88.09^{2+}	88.42
iris	93.07	92.93	93.33	84.80^{3+}	$\mathbf{95.73}^{2-}$	94.80^{1-}	92.94
mammog.-mas.	82.58	82.57	79.94^{3+}	80.42^{3+}	78.58^{3+}	**82.96**	82.56
new-thyroid	94.51	94.05	**95.07**	80.47^{3+}	92.28^{3+}	88.37^{3+}	93.84
parkinsons	79.80	84.00^{2-}	$\mathbf{87.49}^{3-}$	77.03	70.26^{3+}	78.77	77.65
segmentation	**80.80**	**80.80**	74.24^{3+}	73.26^{3+}	76.88^{3+}	77.22^{3+}	80.46
sonar	74.35	75.68	67.70^{3+}	59.92^{3+}	61.19^{3+}	72.99	67.88^{2+}
vehicle	59.50	58.96	58.80^{2+}	56.95^{2+}	57.09^{2+}	59.48	61.76^{2-}
wine	93.49	92.48	95.62^{2-}	79.02^{3+}	96.42^{3-}	96.17^{3-}	93.48
yeast	**52.33**	47.56^{2+}	41.04^{3+}	32.08^{3+}	42.74^{3+}	37.63^{3+}	46.45^{3+}
Average	79.52	79.03	76.13^{1+}	69.60^{3+}	76.14^{2+}	74.30^{1+}	76.84^{2+}
Attained significance level		0.33126	0.03918	0.00021	0.00251	0.01529	0.00985
#OFP.MC better/worse		2/1	10/4	18/1	12/4	11/2	7/3

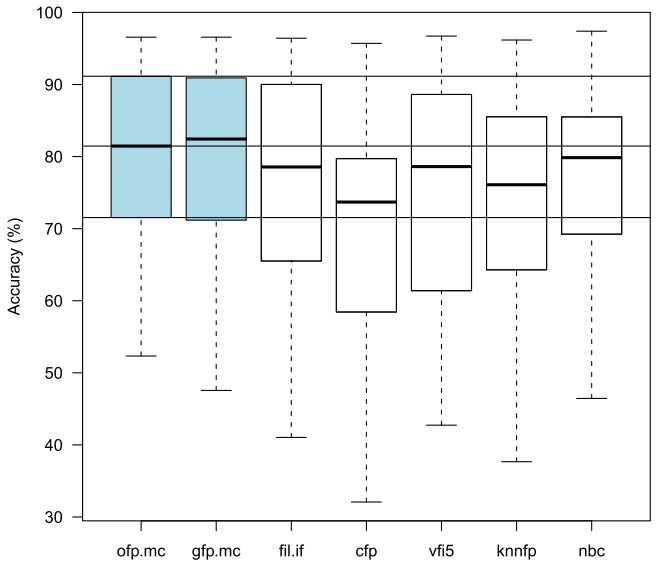

Fig. 10. The boxplots of the average accuracies of the learning algorithms on twenty datasets, reported in Table 4.

Table 4 reports the classification accuracies of the OFP.MC, GFP.MC, FIL.IF, VFI5, CFP, *k*-NNFP, and NBC algorithms. The results are based on the average of the accuracies over five repetitions of 5-fold cross-validation runs. For *k*-NNFP algorithm, optimal *k* is set to an odd integer between 1 and 45 which maximizes the classification accuracy found by 5-fold cross-validation over the training set; ties are broken in favor of the largest values.

For each dataset, the best classification accuracy is shown in boldface. OFP.MC attains the largest average accuracies on 6 datasets, FIL.IF and VFI5 on 4 datasets, NBC on 3 datasets, finally, GFP.MC, CFP and *k*-NNFP on 1 dataset.

To assess the statistical significance of the differences between the accuracies of OFP.MC and other algorithms on every dataset, we perform paired t-tests. If the difference between the accuracies is found statistically significant at 0.05, 0.01, or 0.001 levels, then this is indicated, respectively, by 1±, 2±, or 3± in the superscript attached to the accuracy of the competing algorithm, where + (resp., −) means OFP.MC is better (resp., worse) than the algorithm. No superscript means that the difference between the accuracies of the OFP.MC and the algorithm are statistically insignificant at 0.05 level. The last row counts for each algorithm the number of datasets on which the accuracy of OFP.MC is significantly better/worse than the accuracy of that algorithm. The grand means of average accuracies over twenty datasets of all methods are reported on the row starting with "Average", and the attained significance levels (*p*-values) of the paired t-tests applied to their differences are reported in the following row.

According to paired t-test results, OFP.MC outperforms/underperforms GFP.MC on 2/1, FIL.IF on 10/4, CFP on 18/1, VFI5 on 12/4, *k*-NNFP on 11/2, and NBC on 7/3 out of 20 datasets at some levels of statistical significance. Over twenty datasets, the average accuracy of OFP.MC equals 79.52% and is statistically better than that of FIL.IF and *k*-NNFP at 0.05 significance level, those of VFI5 and NBC at 0.01 significance level, and that of CFP at 0.001 significance level. The difference between the average accuracies of OFP.MC and GFP.MC is statistically insignificant at 0.05 level.

In Fig. 10, the boxplots of the classification accuracies of the learning algorithms on twenty datasets show that OFP.MC and GFP have higher median accuracies than those of all other feature-projection based algorithms. The comparisons of lower and upper quartiles show that the distributions of the average classification accuracies of OFP.MC and GFP.MC are tilted significantly more toward the higher values than those of the other feature-projection based methods. Even though GFP.MC has slightly higher median accuracy than that of OFP.MC, the distribution of the OFP.MC accuracy results concentrates on a range of values slightly higher than that of GFP.MC.

We also examine the sensitivities of the algorithms to the errors in the class boundaries. The plots on the left in Fig. 11 are generated from randomly distributed 55 points on [0,0.5] and [0.5,1] with class labels 0 and 1, respectively. The original data plotted in the top left corner is perfectly separable: all of the class 0 (resp., 1) examples have feature values less (resp., greater) than 0.5. Suppose that five randomly selected points from classes 0 and 1 are moved by mistake to 0.8 and 0.2, respectively, outside their class boundaries, which results in the plot in the bottom left corner. The classification algorithms are run on ten replicas of both noiseless and noisy data generated as above. On the right of Fig. 11 are the boxplots of the classification accuracies of the algorithms run on those ten replicas before and after the noise is introduced. In

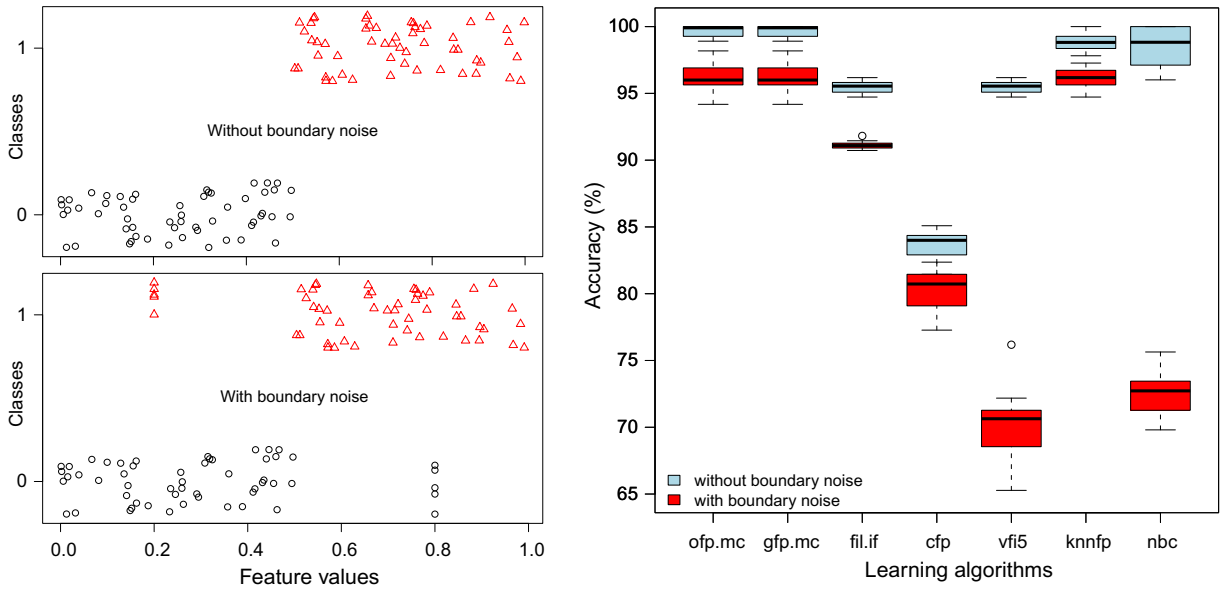

Fig. 11. The sensitivity of the learning algorithms to the noise in class boundaries.

the presence of noise, OFP.MC, GFP.MC, and *k*-NNFP maintain high levels of classification accuracy, but VFI5, FIL.IF, CFP, and NBC algorithms do not. Moreover, the performances of the VFI5 and NBC algorithms are drastically reduced by the noise. Indeed, both boxplots of OFP.MC, GFP.MC, and *k*-NNFP remain relative close to the top line, while the boxplots of other methods are far away either from the top line or from each other.

7. Complexity analysis

We analyze the algorithms in terms of space and time complexities. Time complexity analysis is presented for the training process and for the classification of a single test example. We shall denote by M, N, and K the number of training instances, number of features, and number of classes.

7.1. Space complexity analysis

In the training phase of the OFP.MC algorithm, feature intervals for concept descriptions are constructed on each feature dimension. The space required for storing intervals will be proportional to $M\,N\,K$ at worst case. The space required for storing value and policy functions is proportional to M^2K at worst case. Therefore, the space complexity of the OFP.MC algorithm is $O(M(M+N)K)$.

The space complexities of the FIL.IF and CFP algorithms are $O(M\,N)$, whereas the space complexities of the GFP.MC and VFI5 algorithms are $O(M\,N\,K)$ and $O(N\,K^2)$, respectively.

The *k*-NNFP algorithm stores all instances as feature projections. Therefore, the space required by the *k*-NNFP algorithm is $O(M\,N)$. Because the NBC algorithm computes and stores a distribution for each value on each feature dimension for each class, the space complexity of the NBC algorithm is $O(N\,M\,K)$.

7.2. Time complexity analysis of the training processes

For a dataset with M training instances, feature projections on a feature dimension are sorted with time complexity $O(M\log M)$.

In the OFP.MC algorithm, for each feature, computing value and policy functions using sorted feature projections has time complexity $O(M^2K^2)$ and finding optimal class labels has time complexity $O(M\,K)$. From optimal class labels, feature intervals are constructed in $O(M\,K)$ time. The total complexity of the constrction of intervals on a feature dimenion is $O(M\log M + M^2K^2 + M\,K) = O(M^2K^2)$. Therefore, the overall time complexity of the OFP.MC algorithm for training is $O(N\,M^2K^2)$. However, in general, the number of classes in classification problems is constant. In that case, the time complexity of the OFP.MC algorithm for training is $O(N\,M^2)$.

In the GFP.MC algorithm, for each feature, constructing intervals using sorted feature projections has time complexity $O(M^2K)$. The overall complexity of the construction of intervals on a feature dimension is $O(M\log M + M^2K) = O(M^2K)$. Therefore, the overall time complextiy of the GFP.MC algorithm for training is $O(N\,M^2K)$. If the number of classes is constant, then the time complexity of the GFP.MC for training is $O(N\,M^2)$.

The training time complexities of the FIL.IF and CFP algorithms are $O(N\,M\log M)$ (Dayanik, 2010; Güvenir & Şirin, 1996). The training time complexity of the *k*-NNFP is $O(N\,M\log M)$ (Akkuş & Güvenir, 1996). The training time complexity of the NBC and the VFI5 algorithms is $O(N\,M\,K)$. If the number of classes is constant, then the training time complexity of the NBC and VFI5 algorithm is $O(N\,M)$.

7.3. Time complexity analysis of a single classification process

During the classification process, the function *search-interval*$(f, value)$ finds the interval containing feature value of the test instance on the feature dimension f by a binary search and determines the class votes of that feature. The number of intervals on a feature dimension is at most equal to the number of training instances M. Hence, the worst case time complexity of the search process on a feature dimendion is $O(\log M + K)$ for a feature. Since the final classification is based on the votes of all features, single instance classification time complexity of the OFP.MC and GFP.MC algorithms is $O(N\log M + N\,K) = O(N\log M)$ because the number of classes, K, is usually constant.

The classification time complexities of the FIL.IF, CFP, and *k*-NNFP algorithm is $O(N\log M)$ if $M \gg K$ (Dayanik, 2010; Güvenir & Şirin, 1996; Akkuş & Güvenir, 1996), whereas the classification time complexity of the VFI5 algorithm is $O(N\,K)$. The classification time complexities of the NBC algorithm is $O(N\log M\,K)$.

8. Conclusion

Feature-projection based classification algorithms are simple, yet effective and efficient, produce easy-to-understand concept

descriptions, enable fast classification, and handle missing feature values naturally by simply neglecting them. This paper introduced two new feature-projection based classification algorithms which represent induced classification knowledge as collections of feature intervals. The first algorithm, OFP.MC, optimally partitions each feature using dynamic programming to minimize the feature classification error, while the second algorithm, GFP.MC, greedily partitions each feature to reduce the space requirement of the OFP.MC algorithm.

We showed that the OFP.MC algorithm performs better than the previously proposed feature-projection based classification algorithms on most of the real-world datasets. In the meantime, the new algorithms are insensitive to boundary noise unlike the other feature-projection based classification algorithms considered here.

Although the GFP.MC algorithm is suboptimal in the sense that it does not always find a partition with the smallest possible number of misclassifications on the training set, experimental results indicate that it is competitive with OFP.MC on twenty datasets from the UCI-Repository.

One important future research direction is to learn concept-dependent feature votes for the feature-projection based classification algorithms. Another direction is to investigate better ways to combine predictions of features.

References

Akkuş, A., & Güvenir, H. A. (1996). K nearest neighbor classification on feature projections. In L. Saitta (Ed.), *Proceedings of the 13th international conference on machine learning* (pp. 12–19). Bari, Italy: Morgan Kaufmann.

Asuncion, A., & Newman, D. (2007). UCI machine learning repository.

Chandra, B., & Gupta, M. (2011). Robust approach for estimating probabilities in Naive Bayes classifier for gene expression data. *Expert Systems with Applications: An International Journal Archive, 38*(3).

Chena, J., Huanga, H., Tiana, S., & Qua, Y. (2009). Feature selection for text classification with Naive Bayes. *Expert Systems with Applications, 36*(3), 5432–5435.

Dayanik, A. (2010). Feature interval learning algorithms for classification. *Knowledge-Based Systems, 23*(5), 402–417.

Demiröz, G., & Güvenir, H. (1997). Classification by voting feature intervals. In M. van Someren & G. Widmer (Eds.), *Proceedings of 9th european conference on machine learning, Prague, Czech Republic* (pp. 85–92). Springer.

Domingos, P., & Pazzani, M. (1997). On the optimality of the simple Bayesian classifier under zero-one loss. *Machine Learning, 29*(2-3), 103–130.

Dougherty, J., Kohavi, R., & Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *Proceedings of the International Conference on Machine Learning* (pp. 194–202).

Duda, R. O., Hart, P. E., & Stork, D. G. (2000). *Pattern classification* (second edition). John Wiley & Sons Inc..

Fukunaga, K. (1990). *Introduction to statistical pattern recognition*. Academic Press.

Güvenir, H., & Şirin, I. (1996). Classification by feature partitioning. *Machine Learning, 23*, 47–67.

Güvenir, H., Demiröz, G., & Ilter, N. (1998). Learning differential diagnosis of erythemato-squamous diseases using voting feature intervals. *Artificial Intelligence in Medicine, 13*, 147–165.

Güvenir, H. A., & Emeksiz, N. (2000). An expert system for the differential diagnosis of erythemato-squamous diseases. *Expert Systems With Applications, 18*(1), 43–49.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The weka data mining software: An update. *SIGKDD Explorations, 11*(1).

Hsu, C.-C., Huang, Y.-P., & Chang, K.-W. (2008). Extended Naive Bayes classifier for mixed data. *Expert Systems with Applications, 35*(3), 1080–1083.

Kim, S.-B., Han, K.-S., Rim, H.-C., & Myaeng, S. H. (2006). Some effective techniques for Naive Bayes text classification. *IEEE Transactions on Knowledge and Data Engineering, 18*(11), 1457–1466.

Lewis, D. D. (1998). Naive (Bayes) at forty: The independence assumption in information retrieval. In *Proceedings of the 10th European conference on machine learning* (pp. 4–15). London, UK: Springer.

Liu, H., Hussain, F., Tan, C. L., & Dash, M. (2002). Discretization: An enabling technique. *Data Mining and Knowledge Discovery, 6*(4), 393–423.

Mitchell, T. (1997). *machine learning*. New York: McGraw Hill.

Witten, I. H., & Frank, E. (2005). *Data mining: Practical machine learning tools and techniques* (second edition). Morgan Kaufman.

Yang, Y., & Webb, G. I. (2009). Discretization for Naive–Bayes learning: Managing discretization bias and variance. *Machine Learning, 74*(1), 39–74.